AMENDMENTS TO THE SPECIFICATION

Please amend the paragraph beginning on page 1, line 26 to page 2, line 3 as follows:

In recent years, however, the development man-hours ~~increases~~ have increased by enlargement and diversification of the media processing applications, and application development by a high-level language is also required ~~also~~ in the media processing field. As a result, an attempt is made to realize the development of media processing application by a high-level language. In so doing, the user expects that a more precise tuning can be made even in the case of the development by a high-level language, and therefore it is required to control in detail the optimization strategy performed by the compiler.

Please amend the paragraph beginning on page 2, line 11 as follows:

In view of the foregoing, it is the object of the present invention to provide a highly-flexible compiler such that a user can control optimization by the compiler precisely.

Please amend the paragraph beginning on page 4, line 8 as follows:

Moreover, the compiler according to the present invention receives ~~an~~ a directive on an "if" conversion and performs optimization by the "if" conversion following the directive. As one example, the user can designate not making the "if" conversion by an option at the time of compilation. This makes it possible to prevent a problem that execution of the side with fewer instructions is constrained by the side with more instructions by the "if" conversion from happening when the balance of the number of instructions is not harmonious at a "then" side and an "else" side of an "if" structure.

Please amend the paragraph beginning on page 25, line 7 as follows:

~~Figs.~~Fig. 20 is a diagram showing the format of the instructions executed by the processor 1.

Please amend the paragraph beginning on page 30, line 17 as follows:

The analysis unit 110 transmits to the optimization unit 120 and the output unit 130 directives (options and pragmas) to the compiler 100 and converts a ~~programs~~ program that is an object of the compiler into internal type data by performing lexical analysis on the source program 101 that is an object to be compiled and a directive from a user to this compiler 100.

Please amend the paragraph beginning on page 31, line 3 as follows:

Additionally, "a pragma (or a pragma directive)" is a directive to the compiler 100 that the user can arbitrarily designate (place) in the source program 101 and includes a directive to optimize the code size and the execution time of the generated language program 102 like the option. In the compiler 100 according to the present embodiment, the pragma is a character string starting with "#pragma". For example, the user can describe a statement that #pragma_no_gp_access the name of a variable in the source program 101. This statement is the pragma (the pragma directive). The pragma like this is, different from the option, ~~is~~ treated as an individual directive concerning only the variable placed immediately after the ~~said paramga~~ pragma and loop processing and the like.

Please amend the paragraph beginning on page 31, line 16 as follows:

The optimization unit 120 executes an overall optimization processing to the source program 101 (internal type data) outputted from the analysis unit 110, following a directive from the analysis unit 110 and the like to realize the optimization selected from (1) the optimization with a higher priority on increasing the speed of execution, (2) the optimization with a higher priority on reducing the code size, and (3) the optimization of both of the execution speed and the code size. In addition to this, the optimization unit 120 has a processing unit (a global region allocation unit 121, a software pipelining unit 122, a loop unrolling unit 123, an "if" conversion unit 124, and a pair instruction generation unit ~~125~~ 125) that performs an individual optimization processing designated by the option and the pragma from the user.

Please amend the paragraph beginning on page 34, line 23 as follows:

Note that the one-instruction access range of the gp region is a 14-bit range at the maximum but its range is different based on the type and the size of an object. In other words, an ~~8-bite~~ 8-byte type has a 14-bit range; a ~~4-bite~~ 4-byte type has a 13-bit range; a ~~2-bite~~ 2-byte type has a 12-bit range; and 1-byte type has an 11-bit range.

Please amend the paragraph on page 34, line 28 to page 35, line 2 as follows:

The compiler 100 allocates entities of arrays and structures whose data sizes are smaller than or equal to the maximum data size ~~(32-bite~~ (32-byte default) in the gp region. On the other hand, as for an object that exceeds the maximum data size allocated in the global region, the entity is allocated outside the gp region; in the gp region, only the head address of the object is allocated.

Please amend the paragraph beginning on page 35, line 9 as follows:

Here, NUM is a designated ~~bite~~ byte (32-bit default) of the maximum data size of one array and structure that can be allocated in global region.

Please amend the paragraph beginning on page 36, line 31 to page 37, line 10 as follows:

As is known from the generated codes in the left column of Fig. 40, since the entity of the array C is allocated to a region other than the gp region, it is an absolute address access with plural instructions. On the other hand, as is known from the generated codes in the right column of Fig. 40, the maximum data size (40) is designated in order that the entity of the array C is allocated to the gp region, it is a gp relative access with one-instruction access, and therefore the execution speed increases. In other words, an ~~8-bite~~ 8-byte code executed in 10 cycles is generated by default; a ~~5-bite~~ 5-byte code executed in 7 cycles is generated by changing the maximum data size.

Please amend the paragraph beginning on page 37, line 24 to page 38, line 4 as follows:

On the other hand, as is shown in the right column of Fig. 41, since the defined size of the array A is 40 bytes or less, the entity is allocated to the gp region; a code of the gp relative access with one instruction is generated using the global pointer register (gp). Furthermore, since the size of the externally defined array C is explicitly designated and is smaller than or equal to the maximum data size allocated to the gp region, the entity of the array C is assumed to be allocated to the gp region, the code of the gp relative access is generated. As described above, when the size of the external variable defined outside of the file is not designated, a ~~12-bit~~ 12-byte code executed in 10 cycles is generated; when the maximum data size is changed and the size of the external variable is designated, a ~~5-bite~~ 5-byte code executed in 7 cycles is generated.

Please amend the paragraph beginning on page 40, line 15 as follows:

As described above, when the #pragma _no_gp_access directive is used, a ~~12-bite~~ 12-byte code executed in 10 cycles is generated, while a ~~5-bite~~ 5-byte code executed in 7 cycles is generated when the #pragma _gp_access directive is used.

Please amend the paragraph beginning on page 43, line 6 as follows:

Fig. 44 is a flowchart showing operations of the software ~~pilelining~~ pipelining unit 122. When the option "-fno-software-pipelining" is detected by the analysis unit 110 (Steps S110, S111), the software pipelining unit 122 does not perform the software pipelining optimization to all the loop processing in the source program 101 that is the object (Step S112). Because of this option, it is avoided that the code size increases.